Automatic Differentiation of C and C++ Applications

B. Norris* (ANL), B. Winnicka (U. of Chicago and ANL), P. Hovland (ANL)

Summary

Derivatives, or sensitivities, are ubiquitous in scientific computing. They are used for the solution of inverse problems, including parameter identification and data assimilation, for numerical optimization, and for the solution of nonlinear partial differential equations. Moreover, derivatives give an indication of the sensitivity of simulation outputs to changes in parameters, thus affording insight into physical processes. By providing accurate derivatives with minimal programming effort, automatic differentiation increases the productivity of computational scientists.

Automatic differentiation (AD) tools mechanize the process of developing code for the computation of derivatives. AD avoids the inaccuracies inherent in numerical approximations. Furthermore, sophisticated AD algorithms can often produce code that is more reliable and more efficient than code written by an expert programmer. ADIC is the first and only AD tool for C and C++ based on compiler technology. This compiler foundation makes possible analyses and optimizations not available in tools based on operator overloading. The earliest implementations of ADIC included support for ANSI C applications. Much more complete C++ coverage is available in our recent complete reimplementation (ADIC 2.0), which relies on EDG, a commercial C/C++ parser.

Component AD Tool Infrastructure

Modern AD tools, including ADIC, are implemented in a modular way (see Fig. 1), aiming to isolate language-dependent implementation features from the language-independent program analyses and semantic transformations. The component design leads to much higher implementation quality

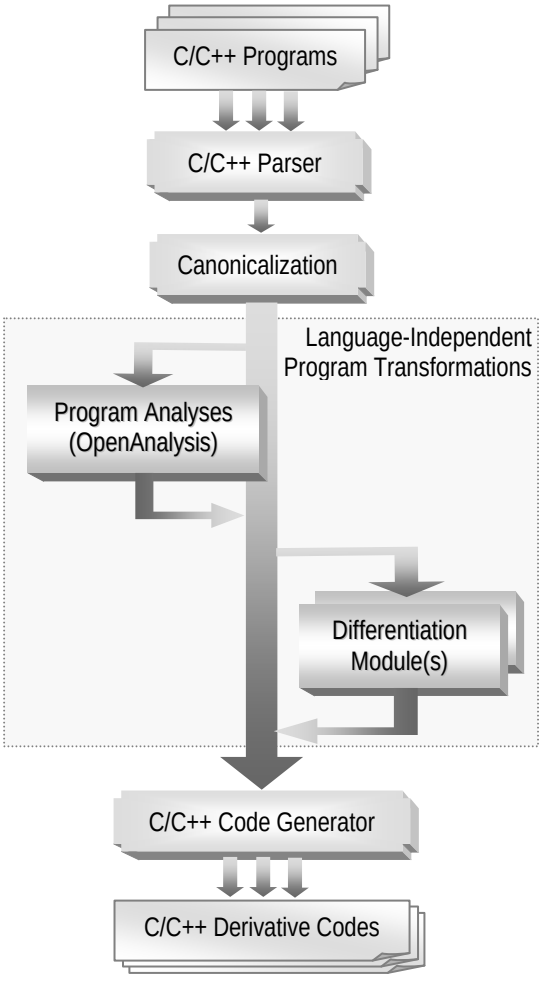

Figure 1. Overview of the AD process and component software infrastructure.

*630-252-7908, norris@mcs.anl.gov

because the different components can be implemented by experts in each of the different domains involved. For example, a compiler expert can focus on parsing, canonicalizing, and unparsing C and C++, while an expert in graph theory and algorithms can produce new differentiation modules without having to worry about the complexity of parsing and generating C++ code. This separation of concerns was achieved through the use of language-independent program analysis interfaces (in collaboration with researchers at Rice University) and a language-independent XML representation of the computational portions of programs. In addition to improved robustness and faster development times, this design naturally enables the reuse of program analysis algorithms and differentiation modules in compiler-based AD tools for other languages. In fact, the analysis and differentiation components are used in both ADIC and the OpenAD/F Fortran front-end (based on Rice's Open64 compiler).

Usability

As the name implies, AD is mostly automatic; however, some manual programming is involved in incorporating the automatically generated differentiated codes into an application. For example, the application developer must specify dependent and independent variables, initialize them correctly, and extract the results from the ADIC-generated derivative objects for further use in the computations. When one is working with arbitrary codes, these manual steps are unavoidable. When the application uses a numerical library, however, there arise opportunities for automating some of these steps. For example, we have made the use of ADIC almost fully automatic in parallel applications that use certain PETSc features for the solution of nonlinear partial differential equations. Our long-term goal is to make the use of automatic differentiation for computing first and second-order derivatives virtually invisible to the user in as many numerical libraries as possible. We have made progress in that direction with the TAO (optimization) and PVODE (ordinary differential equations) toolkits.

ADIC Web Server

To make ADIC more accessible, we have provided a Web-based application server that allows users to invoke ADIC using any browser (see Fig. 2). The user can upload source code, select among various differentiation options, and invoke ADIC by a single button click. The differentiated sources can then be downloaded and compiled locally and linked against the small, portable runtime libraries.

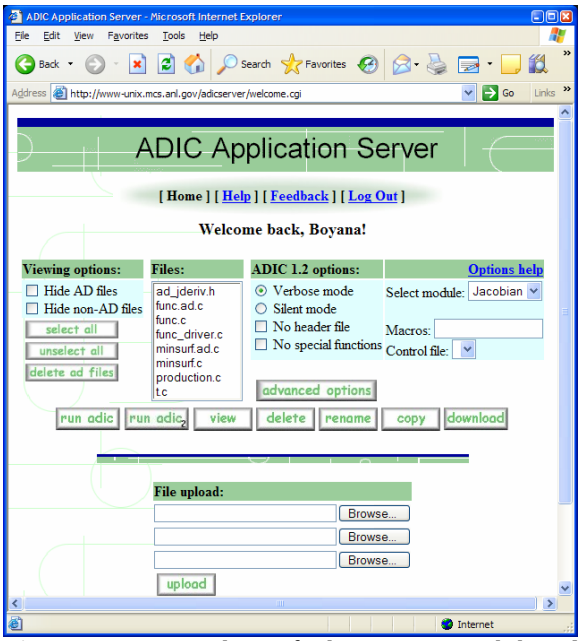

Figure 2. Screenshot of the ADIC Web-based application server.

For further information on this subject contact:
Boyana Norris
Argonne National Laboratory
Mathematics and Computer Science
Argonne, IL 60439
Email: norris@mcs.anl.gov
Phone: 630-252-7908

*630-252-7908, norris@mcs.anl.gov